The TypeScript Tax

A Cost vs Benefit Analysis

Photo: LendingMemo (CC BY 2.0)

TypeScript grew a great deal between 2017 and 2019, and in many ways, for good reason. There's a lot to love about TypeScript. In the 2018 State of JavaScript survey, almost half the respondents said they'd tried TypeScript and would use it again. But should you use it for your large scale app development project?

This article takes a more critical, data-driven approach to analyze the ROI of using TypeScript to build large scale applications.

Translation: 本文采用了一种更关键的、数据驱动的方法来分析使用TypeScript构建大规模应用程序的ROI。

TypeScript Growth

Translation: TypeScript增长

TypeScript is one of the fastest growing languages, and is currently the leading compile-to-JavaScript language.

Google Trends 2014–2019 TypeScript Topic Growth

GitHub Fastest Growing Languages by Contributor Numbers [Source]

This is very impressive traction that shouldn't be discounted, but it is still far from dominating the over-all JavaScript ecosystem. You might say it's a big wave in a much bigger ocean.

Google Search Trends 2014–2018 JavaScript (Red) vs TypeScript (blue) Topic Interest

Translation: 谷歌搜索趋势2014-2018 JavaScript（红色）与TypeScript（蓝色）主题兴趣

GitHub Top Languages by Repositories Created: TypeScript is Not in the Top 5. [Source]

That said, TypeScript hit an inflection point in 2018, and in 2019, a large number of production projects will use it. As a JavaScript developer, you may not have a choice. The

TypeScript decision will be made for you, and you shouldn’t be afraid of learning and using it.

But if you’re in the position of deciding whether or not to use it, you should have a realistic understanding of both the benefits and the costs. Will it have a positive or negative impact?

In my experience, it has both, but falls short of positive ROI. Many developers love using it, and there are many aspects of the TypeScript developer experience I genuinely love. But all of this comes with a cost.

Background

I come from a background using statically typed languages including C/C++ and Java. JavaScript’s dynamic types were hard to adjust to at first, but once I got used to them, it was like coming out of a long, dark tunnel and into the light. There’s a lot to love about static types, but there’s a lot to love about dynamic types, too.

Translation: 我的背景是使用静态类型语言，包括C/C++和Java。JavaScript的动态类型一开始很难适应，但一旦我习惯了它们，就像走出一条漫长而黑暗的隧道，进入光明之中。静态类型有很多值得喜爱的地方，但动态类型也有很多值得喜欢的地方。

On and off over the last few years, I’ve gone all-in on TypeScript full time and racked up more than a year of hands-on daily experience. I went on to lead multiple large-scale production teams using TypeScript as the primary language, and got to see the high-level multi-project impact of TypeScript and compare it to similar large-scale native JavaScript builds.

In 2018, decentralized applications took off, and most of them use smart contracts and open-source software. When you’re dealing with the internet of value, bugs can cost users money. It’s more important than ever to write reliable code, and because these projects are generally open-source, I figured it was nice that we developed the code in TypeScript so that it’s easier for other TypeScript teams to integrate, while maintaining compatibility with projects using JavaScript, as well.

My understanding of TypeScript, including its benefits, costs, and weaknesses have deepened considerably. I’m saddened to say that it wasn’t as successful as I’d hoped. Unless it improves considerably, I would not pick TypeScript for another large scale project.

What I Love About TypeScript

I’m still long-term optimistic about TypeScript. I want to love TypeScript, and there’s a lot I still do love about it. I hope that the TypeScript developers and proponents will read this as a constructive critique rather than a hostile take-down piece. TypeScript developers can fix some of the issues, and if they do, I may repeat the ROI analysis and come to different results.

Static types can be very useful to help document functions, clarify usage, and reduce cognitive overhead. For example, I usually find Haskell’s types to be helpful, low-cost, pain-free, and unobtrusive, but sometimes even Haskell’s flexible higher-kinded type system gets in the way. Try typing a transducer in Haskell (or TypeScript). It’s not easy, and probably a bit worse than the untyped equivalent.

Translation: 静态类型在帮助记录函数、澄清用法和减少认知开销方面非常有用。例如，我通常发现Haskell的类型是有用的、低成本的、无痛的、不引人注目的，但有时甚至Haskell灵活的更善良的类型系统也会成为障碍。尝试在Haskell（或TypeScript）中键入转换器。这并不容易，而且可能比不打字的同类产品更糟糕。

I love that type annotations can be optional in TypeScript when they get in the way, and I love that TypeScript uses structural typing and has some support for type inference (though there’s a lot of room for improvement with inference).

Translation: 我喜欢TypeScript中的类型注释在遇到障碍时可以是可选的，我喜欢TypeScript使用结构类型并支持类型推理（尽管推理还有很大的改进空间）。

TypeScript supports interfaces, which are reusable (as opposed to inline) typings that you can apply in various ways to annotate APIs and function signatures. A single interface can have many implementations. Interfaces are one of the best features of TypeScript, and I wish this feature was built into JavaScript.

The best news: If you use one of the well supported editors (such as Atom or Visual Studio Code), TypeScript’s editor plugins still provide the best IDE developer experience in the JavaScript ecosystem, in my opinion. Other plugin developers should try them out and take notes on how they can improve.

TypeScript ROI in Numbers

I’m going to rate TypeScript on several dimensions on a scale of -10–10 to give you a better sense of how well suited TypeScript may or may not be for large scale applications.

Greater than 0 represents a positive impact. Less than 0 represents a negative impact. 3–5 points represent relatively strong impact. 2 points represents a moderate impact. 1 point represents a relatively low impact.

Translation: 大于0表示积极影响。小于0表示负面影响。3-5分代表相对较强的影响。2分表示中度影响。1分表示相对较低的影响。

These numbers are hard to measure precisely, and will be somewhat subjective, but I've estimated the best I can to reflect the actual costs and rewards we saw on real projects.

Translation: 这些数字很难准确衡量，而且有点主观，但我已经尽我所能估计了真实项目的实际成本和回报。

All projects for which impact was judged were >50k LOC with several collaborators working over several months. One project was Angular 2 + TypeScript, compared against a similar project written in Angular 1 with standard JavaScript. All other projects were built with React and Node, and compared against React/Node projects written in standard JavaScript. Subjective bug density, subjective relative velocity, and developer feedback were estimated, but not precisely measured. All teams contained a mix of experienced and new TypeScript developers. All members had access to more experienced mentors to assist with TypeScript onboarding.

Objective data was too noisy in the small sampling of projects to make any definitive objective judgements with a reliable error margin. On one project, native JavaScript showed a 41% lower public bug density over TypeScript. In another, the TypeScript project showed a 4% lower bug density over the comparable native JavaScript version. Obviously, the implementation (or lack) of other quality measures had a much stronger effect than TypeScript, which skewed the numbers beyond usability.

With margin-of-error so broad, I gave up on objective quantification, and instead focused on feature delivery pace and observations of where we spent our time. You'll see more of those details in the ROI point-by-point breakdown.

Because there's a lot of subjectivity involved, you should allow for a margin of error in interpretation (pictured in the chart), but the over-all ROI balance should give you a good idea of what to expect.

TypeScript Cost vs Benefit Analysis: Likely Negative ROI

Translation: TypeScript成本与效益分析：可能的负ROI

I can already hear the peanut gallery objections to the small benefits scores, and I don't entirely disagree with the arguments. TypeScript does provide some very useful, powerful capabilities. There's no question about that.

In order to understand the relatively small benefit scores, you have to have a good understanding of what I'm comparing TypeScript to: Not just JavaScript, but JavaScript paired with tools built for native JavaScript.

Let's look at each point in more detail.

Developer Tooling: My favorite feature of TypeScript, and arguably the most powerful practical benefit from using TypeScript is its ability to reduce the cognitive load of developers by providing interface type hints and catch potential errors in realtime as you're programming. If none of that were possible in native JavaScript with some good plugins, I'd give TypeScript more points on the benefit side, but the 0 point is what's already available using JavaScript, and the baseline is already pretty good.

Most TypeScript advocates don't seem to have a good understanding of what TypeScript is competing against. The development tool choice isn't TypeScript vs native JavaScript and no tooling. It's between TypeScript and the entire rich ecosystem of JavaScript developer tools. Native JavaScript autocomplete and error detection gets you 80% — 90% of the benefits of TypeScript when you use autocomplete, type inference, and lint tooling. When you're running type inference, and you use ES6 default parameters, you get type hints just like you would with type-annotated TypeScript code.

Example of Native JavaScript Autocomplete with Type Inference

Translation: 带有类型推断的本地JavaScript自动完成示例

In fairness, if you use default parameters to provide type hints, you don't need to supply the annotations for TypeScript code, either, which is a great trick to reduce type syntax overhead — one of the overhead costs of using TypeScript.

TypeScript's tooling for these things is arguably a little better, and more all-in-one — but it's not enough of an improvement to justify the costs.

API Documentation: Another great benefit of TypeScript is better documentation for APIs which is always in sync with your source code. You can even generate API documentation from your TypeScript code. This would also get a higher score, except you can get the same benefit using JSDoc and Tern.js in JavaScript, and documentation generators are abundant. Personally, I'm not a big fan of JSDoc, so TypeScript does get some points, here.

Even with the best inline documentation in the world, you still need real documentation, so TypeScript enhances, rather than replaces existing documentation options.

Refactoring. In most cases, if you can gain a significant benefit from TypeScript in your refactoring, that's often a code smell indicating that your code is too tightly coupled. I have written an entire book on how to write more composable, more loosely coupled code, called "Composing Software". If TypeScript is saving you a lot of refactoring pain, there's a good chance tight coupling is still causing you a lot of other avoidable problems. I strongly suggest reading the book, particularly the chapter "Mocking is a Code Smell", which provides a lot of information on the causes of tight coupling and some best practices that can help you avoid them.

On the other hand, some companies run very large ecosystems of connected projects sharing the same code repository (e.g., Google's famous monorepo). Using TypeScript enables them to upgrade API design choices to account for better designs and new use-cases. The developers responsible for those upgrades are also responsible for ensuring that their library changes don't break any of the software in the monorepo that depends on those libraries. TypeScript may offer significant time savings for this very limited subset of TypeScript users.

Translation: 另一方面，一些公司运行着共享同一代码库的大型互联项目生态系统（例如，谷歌著名的monoreo）。使用TypeScript使他们能够升级API设计选项，以实现更好的设计和新的用例。负责这些升级的开发人员还负责确保他们的库更改不会破坏monoreto中依赖于这些库的任何软件。TypeScript可以为这个非常有限的TypeScript用户子集节省大量时间。

I say very limited subset, because giant, closed monorepo ecosystems are the exception, rather than the rule. The process might scale across Google, but can't scale to repositories that the library authors are not aware of. Making breaking changes to library APIs used by a broader ecosystem can break code you don't even know exists.

Translation: 我说的是非常有限的子集，因为巨大的、封闭的单回购生态系统是例外，而不是规则。这个过程可能会扩展到整个谷歌，但不能扩展到图书馆作者不知道的存储库。对更广泛的生态系统使用的库API进行突破性的更改可能会破坏你甚至不知道存在的代码。

In traditional, more decentralized library ecosystems, people avoid breaking changes to APIs, and instead create new features following the open/closed principle (APIs are open for extension, and closed to breaking changes). This is how the web platform itself has mostly evolved, with a few exceptions. This is why React still supports features that have been replaced by better options since React 0.14. React evolves and adds great new features, radically improving the developer experience without breaking old functionality. For instance, class components will still be supported by React, even after the much improved React Hooks API matures.

That makes changes across the whole ecosystem optional, rather than required. Teams can upgrade their software gradually, on an as-needed basis rather than heaping a whole-ecosystem code change project on the library team.

Even in cases where whole ecosystem code changes are required, type inference and automated codemods can help — no TypeScript required.

I initially mentally scored refactoring a zero and left it off the list because I strongly favor the open/closed approach, inference, and codemods. However, some teams are getting real benefits from it under limited circumstances.

Translation: 我最初在心理上给重构打分为零，并将其从列表中删除，因为我强烈支持开放/封闭方法、推理和代码模式。然而，一些球队在有限的情况下从中获得了真正的好处。

There's a very good chance that you'd be better served in other ways using native JavaScript.

Translation: 很有可能使用本机JavaScript以其他方式为您提供更好的服务。

Type safety doesn't seem to make a big difference. TypeScript proponents frequently talk about the benefits of type safety, but there is little evidence that type safety makes much difference (really, static types seem to have very little impact) to production bug density. (Even more evidence that TypeScript does not have a big impact on bug reduction from a 2022 study). This is important because code review and TDD make a very big difference (40% — 80% for TDD alone). Pair TDD with design review, spec review, and code review, and you're looking at 90%+ reductions in bug density. Many of those processes (particularly TDD) are capable of catching all of the same class of bugs that TypeScript catches, as well as many bugs that TypeScript will never be able to catch.

TypeScript is only capable of addressing about 20% of "public bugs", where public means that the bugs survived past the implementation phase and got committed to the public repository, according to Zheng Gao and Earl T. Barr from University College London, and Christian Bird from Microsoft Research.

The authors of this study think they've underestimated the impact of TypeScript because they assume that all the other quality measures have already been applied, but they made no effort to judge the quality of the other bug prevention measures. They acknowledge the variable, but leave it entirely out of the calculations.

In my experience, the vast majority of teams have partially applied some measures, but rarely applied all important bug prevention measures well. On my teams, we use design review, spec review, TDD, code review, lint, schema validation, and company-sponsored mentorship, which all have dramatic impacts on bug density, reducing type errors to very near zero.

In my experience, all but linting have a larger impact on code quality than static types. In other words, I'm starting from a much stricter definition of zero than the authors of the paper.

Translation: 根据我的经验，除了linting之外，其他所有类型对代码质量的影响都比静态类型大。换句话说，我从一个比论文作者更严格的零定义开始。

If you have not properly implemented those other bug prevention measures, I have no doubt you'll see 15% — 18% reduction in bug density using TypeScript alone, but you'll also completely miss 80% of the bugs until they get to production and start causing real problems.

Translation: 如果你没有正确地实施这些其他的错误预防措施，我毫不怀疑你会看到仅使用TypeScript就可以减少15%-18%的错误密度，但你也会完全错过80%的错误，直到它们进入生产并开始引起真正的问题。

Some will argue that TypeScript provides realtime bug feedback, so you can catch the bugs earlier, but so do type inference, lint, and TDD (I set up a watch script to run my unit tests on file save, so I get very near immediate, rich feedback). You may argue that these other measures have a cost, but because TypeScript will always miss 80% of bugs, you can't safely skip them either way, so their cost applies to both sides of the ROI math, and is already factored in.

Translation: 有些人会争辩说，TypeScript提供了实时的错误反馈，所以你可以更早地发现错误，但类型推理、lint和TDD也是如此（我设置了一个观察脚本来在文件保存时运行我的单元测试，所以我得到了非常即时、丰富的反馈）。你可能会争辩说，这些其他措施是有成本的，但由于TypeScript总是会遗漏80%的错误，你无法安全地跳过它们，因此它们的成本适用于ROI数学的两面，并且已经被考虑在内。

The study looked at bugs that were known in advance, including the exact lines that were changed to fix the bugs in question, where the problem and potential solutions were known prior to introduction of typings. What this means is that even knowing that the bugs existed in advance, TypeScript was unable to detect 85% of public bugs — catching only 15%.

Update: We’re going to give TypeScript a generous benefit of the doubt and use 20% in our calculations, to drive home the point about exponentially diminishing returns.

Why are so many bugs undetectable by TypeScript? For starters, specification errors caused about 78% of the publicly classified bugs studied on GitHub. The failure to correctly specify behaviors or correctly implement a specification is the most common type of bug by a huge margin, and that fact automatically renders an overwhelming majority of bugs impossible for TypeScript to detect or prevent. In “To Type or Not to Type”, the study authors identified and classified a range of “ts-undetectable” bugs.

Translation: 为什么TypeScript检测不到这么多错误？首先，在GitHub上研究的公开分类错误中，约78%是由规范错误引起的。未能正确指定行为或正确实现规范是最常见的错误类型，这一事实自动导致TypeScript无法检测或预防绝大多数错误。在“打字还是不打字”中，研究作者确定并分类了一系列“无法检测”的错误。

Histogram of ts-undetectable bugs. Source: “To Type or Not to Type”

Translation: 无法检测到的错误的直方图。来源：“要键入还是不键入”

“StringError” above are the classification of errors where the string was the right type, but contained the wrong value (like an incorrect URL). Branch errors and predicate errors are logic errors that led to the wrong code paths being used. As you can see there are a variety of other errors that TypeScript just can’t touch. There’s little potential that TypeScript will ever be capable of detecting more than 20% of bugs.

Translation: 上面的“StringError”是错误的分类，其中字符串是正确的类型，但包含错误的值（如不正确的URL）。分支错误和谓词错误是导致使用错误代码路径的逻辑错误。正如您所看到的，还有许多其他TypeScript无法触及的错误。TypeScript能够检测到20%以上的错误的可能性很小。

But a 20% sounds like a lot! Why doesn’t TypeScript get much higher bug prevention points?

Because there are so many bugs that are not detectable by static types, it would be irresponsible to skip other quality control measures like design review, spec review, code review, and TDD. So it’s not fair to assume that TypeScript will be the only thing you’re employing to prevent bugs. In order to really get a sense of ROI, we have to apply the bug reduction math after discounting the bugs caught by other measures which were not adequately factored in by the study authors.

It’s unsafe to skip other measures: Spec errors: 80% — Type errors: 20%

Imagine your project would have contained 1,000 bugs with no bug prevention measures. After applying other quality measures, the potential production bug count is reduced to 100. Now we can look at how many additional bugs TypeScript would have prevented to get a truer sense of the bug catching return on our TypeScript investment. Close to 80% of bugs are not detectable by TypeScript, and all TypeScript-detectable bugs can potentially be caught with other measures like TDD.

Translation: 想象一下，如果没有错误预防措施，你的项目将包含1000个错误。在应用其他质量措施后，潜在的生产错误数将减少到100。现在，我们可以看看TypeScript可以防止多少额外的bug，以更真实地了解我们的TypeScript投资的bug捕获回报。接近80%的错误是TypeScript无法检测到的，所有TypeScript可检测到的错误都可能被TDD等其他措施捕获。

No measures: 1000 bugs

Translation: 无措施：1000个错误

After other measures: 100 bugs remain — 900 bugs caught

Translation: 采取其他措施后：仍有100个漏洞——900个漏洞被捕获

After adding TypeScript to other measures: 80 bugs remain — 20 more bugs caught

Translation: 在将TypeScript添加到其他度量中后：仍有80个错误——又捕获了20个错误

Bar graph of bugs remaining after applying reduction measures: TypeScript provides little added benefit.

Translation: 应用减少措施后剩余bug的条形图：TypeScript几乎没有提供额外的好处。

Some people argue that if you have static types, you don’t need to worry about writing so many tests. Those people are making a silly argument. There is really no contest. Even if you’re going to employ TypeScript, you still need the other measures.

Translation: 有些人认为，如果你有静态类型，你就不需要担心写那么多测试。那些人在进行愚蠢的争论。真的没有竞争。即使你要使用TypeScript，你仍然需要其他措施。

Chart: Adding TypeScript after reviews, TDD catches a tiny fraction of the total bug count.

Translation: 图表：在审查后添加TypeScript，TDD只捕获了总错误数的一小部分。

In this scenario, reviews and TDD catch 900/1,000 bugs without TypeScript. TypeScript catches 200/1,000 bugs if you skip reviews and TDD. You obviously don’t have to pick one or the other, but adding TypeScript after applying other measures leads to a very small improvement due to exponentially diminishing returns.

Translation: 在这个场景中，审查和TDD在没有TypeScript的情况下捕获900/1000个错误。如果你跳过评论和TDD，TypeScript会捕获200/1000个错误。显然，您不必选择其中一个，但在应用其他措施后添加TypeScript会导致非常小的改进，因为回报呈指数级递减。

Update: 2019–02–11:

Translation: 更新：2019年2月11日：

Airbnb recently reported a 38% reduction in bugs by adding TypeScript to their development process. How could that be? According to this article, that should be impossible, right? That’s not how the math works. We’re dealing with percentages, averages, and diminishing returns, not concrete values.

Translation: Airbnb最近报告称，通过在其开发过程中添加TypeScript，漏洞减少了38%。怎么可能呢？根据这篇文章，这应该是不可能的，对吧？数学不是这样运作的。我们处理的是百分比、平均值和递减收益，而不是具体的价值。

The study this article relies on represents averages, and the presence or absence of other quality measures impacts the percentage of bugs remaining that TypeScript could address.

Translation: 本文所依赖的研究代表了平均值，其他质量指标的存在与否会影响TypeScript可以解决的剩余错误的百分比。

The more ts-undetectable bugs the other measures address, the higher the percentage of remaining bugs TypeScript can address, but those other measures also reduce the total number of remaining bugs for TypeScript to address. So the percentage might go up, but the total number of bugs caught might change only a little.

Translation: 其他措施解决的无法检测的错误越多，TypeScript可以解决的剩余错误百分比就越高，但这些其他措施也会减少TypeScript要解决的剩余缺陷总数。因此，百分比可能会上升，但被捕获的错误总数可能只会发生一点变化。

Also, careful code review does a great job of catching and reducing bugs, and there's no more careful code review than a complete overhaul of the entire codebase, carefully inspecting and analyzing every line of code. I'd expect a ~30% reduction in bugs just from that act, alone, (even if they left it in JavaScript) regardless of types.

Translation: 此外，仔细的代码审查在捕捉和减少错误方面做得很好，没有什么比对整个代码库进行彻底检查、仔细检查和分析每一行代码更仔细的代码检查了。我预计，仅从这一行为开始，无论类型如何，bug都会减少约30%（即使他们把它留在JavaScript中）。

As of this writing, they have not released their methodology or reported what other bug reduction measures they're employing, but my guess is that they're employing some form of design/spec review process to reduce the share of specification bugs that make it into their code in the first place.

Translation: 截至本文撰写之时，他们还没有发布他们的方法，也没有报告他们正在采用的其他减少错误的措施，但我猜他们正在采用某种形式的设计/规范审查过程，以减少最初出现在代码中的规范错误。

In other words, when you eliminate a lot of bugs that TypeScript can't help with, TypeScript can provide a higher percentage of bug reduction to the remaining bugs.

Translation: 换句话说，当你消除了很多TypeScript无法帮助的bug时，TypeScript可以为剩余的bug提供更高比例的bug减少。

This result doesn't change the fact that only 20% of public bugs can even be addressed by TypeScript, and doesn't invalidate the point about exponentially diminishing returns.

Translation: 这一结果并没有改变这样一个事实，即只有20%的公共漏洞可以通过TypeScript来解决，也没有使收益呈指数递减的观点无效。

Instead, it implies that Airbnb may have better-than-average design or spec review (or both), coupled with lower-than-average automated code coverage — perhaps missing unit test coverage, functional test coverage, or both. Proper unit test coverage can catch close to 100% of the bugs that static types can catch, along with a lot of bugs TypeScript can't catch.

Translation: 相反，这意味着Airbnb的设计或规范审查（或两者兼而有之）可能优于平均水平，同时自动化代码覆盖率也低于平均水平——可能缺少单元测试覆盖率、功能测试覆盖率，或两者兼而有之。适当的单元测试覆盖率可以捕获静态类型可以捕获的几乎100%的错误，以及TypeScript无法捕获的许多错误。

Most teams have little or no design/spec review process implemented. Even having an engineer look at mock-ups with a critical eye before handing them off to a developer to implement would be better than average. Many teams don't have any formal design review process at all.

Translation: 大多数团队很少或根本没有实施设计/规范审查流程。即使在将模型交给开发人员实施之前，让工程师以批判的眼光看待模型，也比平均水平要好。许多团队根本没有任何正式的设计审查流程。

Here's what their TypeScript benefit chart might look like:

Translation: 以下是他们的TypeScript福利图表：

TypeScript is still catching just 38 out of 1,000 potential bugs, but since most of the potential bugs are caught by previous steps in the pipeline (like people reviewing mockups before they go to a developer to implement), TypeScript can address a larger share of remaining bugs. In this case, 18 more bugs than teams missing Airbnb's additional code quality measures.

Translation: TypeScript仍然只捕获了1000个潜在错误中的38个，但由于大多数潜在错误都是通过之前的步骤捕获的（比如人们在去开发人员那里实现之前审查模型），因此TypeScript可以解决更大份额的剩余错误。在这种情况下，比错过Airbnb额外代码质量措施的团队多出18个bug。

The diminishing returns math could only be completely invalidated if TypeScript could catch a much larger share of all bugs: closer to 75%+, because at that stage, it might be viable to replace other expensive parts of the quality control process, like code review or TDD.

Translation: 只有当TypeScript能够捕捉到更大份额的所有错误时，收益递减数学才能完全无效：接近75%+，因为在那个阶段，替换质量控制过程中其他昂贵的部分（如代码审查或TDD）可能是可行的。

It would be interesting to learn exactly how many bugs Airbnb caught during the conversion to TypeScript, to learn about the classification of the bugs that TypeScript couldn't prevent, to learn the bug density (and how they calculated it), and to learn what other quality control measures they already employ.

Translation: 了解Airbnb在转换为TypeScript的过程中发现了多少漏洞，了解TypeScript无法防止的漏洞的分类，了解漏洞密度（以及他们是如何计算的），以及了解他们已经采用

的其他质量控制措施，将是一件有趣的事。

Keep in mind: I’m not arguing against using TypeScript. I’m arguing for people to consider the costs and benefits and make a rational, informed decision that’s right for you and your team. Some products require stricter quality control, and it may be worth the extra cost to eliminate 18 more bugs out of a thousand. For example, if your code powers critical parts of the self-driving system for a Tesla, I hope you’re using static types along with all the other quality measures, because the cost of bugs is much higher. Each team should conduct their own ROI analysis and make the decision that is right for them.

Translation: 请记住：我并不是反对使用TypeScript。我主张人们考虑成本和收益，做出合理、明智的决定，这对你和你的团队来说是正确的。有些产品需要更严格的质量控制，在一千个缺陷中再消除18个可能是值得的。例如，如果你的代码为特斯拉自动驾驶系统的关键部分供电，我希望你在使用静态类型和所有其他质量指标的同时，使用静态类型，因为错误的成本要高得多。每个团队都应该进行自己的ROI分析，并做出适合自己的决定。

Having implemented quality control systems on large scale, multi-million dollar development projects, I can tell you that my expectations for effectiveness on costly system implementations are in the territory of 30% — 80% reductions. You can get those kinds of numbers from any of the following:

Translation: 在大规模、数百万美元的开发项目中实施了质量控制系统后，我可以告诉你，我对成本高昂的系统实施的有效性的期望是降低30%-80%。您可以从以下任意一个中获得这些类型的数字：

Design and Spec Review (up to 80% reduction)

Translation: 设计和规范审查（最多减少80%）

TDD (40% — 80% reduction of remaining bugs)

Translation: TDD（剩余错误减少40%-80%）

Code Review (an hour of code review saves 33 hours maintenance)

Translation: 代码审查（一小时的代码审查可节省33小时的维护时间）

It turns out that type errors are just a small subset of the full range of possible bugs, and there are other ways to catch type errors. The data is in, and the result is very clear: TypeScript won’t save you from bugs. At best, you’ll get a very modest reduction, and you still need all your other quality measures.

Translation: 事实证明，类型错误只是所有可能错误中的一小部分，还有其他方法可以捕捉类型错误。数据在中，结果非常清楚：TypeScript不会让你免于bug。充其量，你会得到一个非常温和的削减，而且你仍然需要所有其他的质量指标。

Type correctness does not guarantee program correctness.

Translation: 类型正确性不能保证程序的正确性。

It looks like the benefits are not living up to the TypeScript hype. But those can't be the only benefits, right?

Translation: 看起来这些好处并没有达到TypeScript的宣传效果。但这些并不是唯一的好处，对吧?

New JavaScript Features and Compile to Cross-Browser JavaScript: Babel does both for native JavaScript.

Translation: 新的JavaScript功能和编译到跨浏览器JavaScript：Babel为原生JavaScript做这两件事。

We've reached the end of the benefits, and I don't know about you, but I'm feeling a little underwhelmed. If we can get type hints, autocomplete, and great bug reductions for native JavaScript using other tools, the only question that remains is, does the TypeScript difference pay off the investment required to use it?

Translation: 我们已经到了福利的尽头，我不知道你的情况，但我感到有点失望。如果我们可以使用其他工具为原生JavaScript提供类型提示、自动完成和极大的bug减少，那么剩下的唯一问题是，TypeScript的差异是否能弥补使用它所需的投资?

To figure that out, we need to take a closer look at the costs of TypeScript.

Translation: 要弄清楚这一点，我们需要仔细研究TypeScript的成本。

Recruiting: While nearly half of The State of JavaScript respondents have used TypeScript and would use it again, and an additional 33.7% would like to learn, 5.4% have used TypeScript and would not use it again, and 13.7% are not interested in learning TypeScript. That reduces the recruiting pool by almost 20%, which could be a significant cost to teams who need to do a lot of hiring. Hiring is an expensive process which can drag on for months and cut into the productive time of your other developers (who, more often than not, are the people most qualified to assess new candidate's skills).

Translation: 招聘：虽然近一半的《JavaScript现状》受访者曾使用过TypeScript并将再次使用，另有33.7%的人想学习，5.4%的人使用过TypeScript并不会再次使用，13.7%的人对学习TypeScript不感兴趣。这将使招聘队伍减少近20%，这对需要大量招聘的团队来说可能是一笔巨大的成本。招聘是一个昂贵的过程，可能会拖上几个月，并占用其他开发人员的生产时间（他们往往是最有资格评估新候选人技能的人）。

On the other hand, if you only need to hire one or two developers, using TypeScript may make your opening more attractive to almost half the candidate pool. For small projects, it may be a wash, or even slightly positive. For teams of hundreds or thousands, it’s going to swing into the negative side of the ROI error margin.

Translation: 另一方面，如果你只需要雇佣一两名开发人员，使用TypeScript可能会让你的职位对几乎一半的候选人更有吸引力。对于小项目来说，这可能是一种洗白，甚至是轻微的积极影响。对于成百上千的团队来说，它将进入ROI误差范围的负值。

Setup, Initial Training: Because these are one-time costs, they’re relatively low. Teams already familiar with JavaScript tend to get productive in TypeScript within 2–3 months, and pretty fluent within 6–8 months. Definitely more costly than recruiting, but certainly worth the effort if this were the only cost.

Translation: 设置、初始培训：因为这些都是一次性费用，所以相对较低。已经熟悉JavaScript的团队往往在2-3个月内就能熟练使用TypeScript，在6-8个月内就会非常流利。这肯定比招聘成本更高，但如果这是唯一的成本，那肯定是值得的。

Missing Features — HOFs, Composition, Generics with Higher Kinded Types, Etc.: TypeScript is not fully coexpressive with idiomatic JavaScript. This is one of my biggest challenges (and expenses) with TypeScript, because fluent JavaScript developers will frequently encounter situations which are difficult or impossible to type, but conscientious developers will be interested in doing things right. They’ll spend hours Googling for examples, trying to learn how to type things that TypeScript simply can’t type properly.

Translation: 缺少的功能--HOF、Composition、具有更高级类型的泛型等。TypeScript与惯用JavaScript不能完全共表达。这是我在使用TypeScript时面临的最大挑战（和开支）之一，因为流利的JavaScript开发人员经常会遇到难以或不可能打字的情况，但有良知的开发人员会对做好事情感兴趣。他们会花几个小时在谷歌上搜索示例，试图学习如何键入TypeScript根本无法正确键入的内容。

TypeScript could improve on this cost by providing better documentation and discovery of TypeScript’s current limitations, so developers waste less time trying to get it to behave well on higher order functions, declarative function compositions, transducers, and so on.

In many cases, a well-behaved, readable, maintainable TypeScript typing simply isn’t going to happen. Developers need to be able to discover that quickly so that they can spend their time on more productive things.

Translation: TypeScript可以通过提供更好的文档和发现TypeScript当前的局限性来提高这一成本，因此开发人员可以减少浪费时间，让它在高阶函数、声明性函数组合、转换器等方面表现良好。在许多情况下，表现良好、可读、可维护的TypeScript类型根本不会发生。开发人员需要能够快速发现这一点，以便将时间花在更高效的事情上。

Ongoing Mentorship: While people get productive with TypeScript pretty quickly, it does take quite a bit longer to get feeling confident. I still feel like there’s a lot more to learn. In TypeScript, there are different ways to type the same things, and figuring out the advantages and disadvantages of each, teasing out best practices, etc. takes quite a bit longer than the initial learning curve.

Translation: 持续导师制：虽然人们很快就能通过TypeScript提高工作效率，但确实需要更长的时间才能感到自信。我仍然觉得还有很多东西需要学习。在TypeScript中，有不同的方法来键入相同的东西，找出每种方法的优缺点、找出最佳实践等需要比最初的学习曲线更长的时间。

For example, new TypeScript developers tend to over-use annotations and inline typings, while more experienced TypeScript developers have learned to reuse interfaces and create separate typings to reduce the syntax clutter of inline annotations. More experienced developers will also spot ways to tighten up the typings to produce better errors at compile time.

Translation: 例如，新的TypeScript开发人员倾向于过度使用注释和内联打字，而更有经验的TypeScript开发者已经学会了重用接口和创建单独的打字，以减少内联注释的语法混乱。更有经验的开发人员还将找到加强打字的方法，以便在编译时产生更好的错误。

This extra attention to typings is an ongoing cost you’ll see every time you onboard new developers, but also as your experienced TypeScript developers learn and share new tricks with the rest of the team. This kind of ongoing mentorship is just a normal side-effect of collaboration, and it’s a healthy habit that saves money in the long term when applied to other things, but it comes at a cost, and TypeScript adds significantly to it.

Translation: 这种对打字的额外关注是一种持续的成本，每次你加入新的开发人员时，你都会看到这种成本，但当你有经验的TypeScript开发人员学习并与团队其他成员分享新技巧时，也是如此。这种持续的指导只是合作的一种正常副作用，从长远来看，这是一种健康

的习惯，当应用于其他事情时可以节省资金，但这是有代价的，TypeScript大大增加了成本。

Typing Overhead: In the cost of typing overhead, I’m including all the extra time spent typing, testing, debugging, and maintaining type annotations. Debugging types is a cost that is often overlooked. Type annotations come with their own class of bugs. Typings that are too strict, too relaxed, or just wrong.

Translation: 打字开销：在打字开销中，我包括了花在打字、测试、调试和维护类型注释上的所有额外时间。调试类型是一个经常被忽视的成本。类型注释自带一类bug。打字过于严格、过于放松或只是错误。

This cost center has gone down since I first explored it, because many third party libraries now contain typings, so you don’t have to do so much work trying to track them down or create them yourself. However, many of those typings are still broken and out-of-date in all but the most popular OSS packages, so you’ll still end up backfilling typings for third party libraries that you want type hints for. Often, developers try to get those typings added upstream, with widely varied results.

Translation: 自从我第一次探索这个成本中心以来，它就一直在下降，因为许多第三方库现在都包含打字员，所以你不必做那么多工作来追踪或自己创建它们。然而，除了最流行的OSS软件包之外，这些打字法中的许多仍然是坏的和过时的，所以你最终仍然会为你想要类型提示的第三方库回填打字法。通常，开发人员试图将这些打字法添加到上游，结果千差万别。

You may also notice greatly increased syntax noise. In languages like Haskell, typings are generally short one-liners listed above the function being defined. In TypeScript, particularly for generic functions, they’re often intrusive and defined inline by default.

Translation: 您可能还会注意到语法噪音大大增加。在像Haskell这样的语言中，打字通常是在所定义的函数上方列出的简短的一行代码。在TypeScript中，特别是对于泛型函数，它们通常是侵入性的，默认情况下是内联定义的。

Instead of adding to the readability of a function signature, TypeScript typings can often make them harder to read and understand. This is one reason experienced TypeScript developers tend to use more reusable typings and interfaces, and declare typings separately from function implementations. Large TypeScript projects tend to develop their own libraries of reusable typings that can be imported and used anywhere in the project, and maintenance of those libraries can become an extra — but worthwhile — chore.

Translation: TypeScript打字通常会使函数签名更难阅读和理解，而不是增加函数签名的可读性。这就是经验丰富的TypeScript开发人员倾向于使用更可重用的打字法和接口，并将打字法与函数实现分开声明的原因之一。大型TypeScript项目倾向于开发自己的可重复使用的打字库，这些打字库可以在项目中的任何地方导入和使用，维护这些库可能会成为一项额外但有价值的工作。

Syntax noise is problematic for several reasons. You want to keep your code free of clutter for the same reasons you want to keep your house free of clutter:

Translation: 语法噪声是有问题的，原因有几个。你想让你的代码没有杂乱，原因与你想让房子没有杂乱的原因相同：

More clutter = more places for bugs to hide = more bugs.

Translation: 更多的混乱=隐藏bug的地方越多=bug越多。

More clutter makes it harder to find the information you’re looking for.

Translation: 更多的混乱会使你更难找到你想要的信息。

Clutter is like static on a poorly tuned radio — more noise than signal. When you eliminate the noise, you can hear the signal better. Reducing syntax noise is like tuning the radio to the proper frequency: The meaning comes through more easily.

Translation: 杂波就像调谐不好的收音机上的静电——噪音比信号大。当你消除噪音时，你可以更好地听到信号。减少语法噪音就像将收音机调到合适的频率：意思更容易理解。

Syntax noise is one of the heavier costs of TypeScript, and it could be improved on in a couple ways:

Translation: 语法噪声是TypeScript较重的成本之一，可以通过以下几种方式进行改进：

Better support for generics using higher-kinded types, which can eliminate some of the template syntax noise. (See Haskell’s type system for reference).

Translation: 更好地支持使用更高级类型的泛型，这可以消除一些模板语法噪声。（参考Haskell的类型系统）。

Encourage separate, rather than inline typings, by default. If it became a best practice to avoid inline typings, the typing syntax would be isolated from the function implementation, which would make it easier to read both the type signature and the implementation, because they wouldn’t be competing with each other. This could be implemented as a documentation overhaul, along with some evangelism on Stack Overflow.

Translation: 默认情况下，鼓励单独打字，而不是内联打字。如果避免内联打字成为最佳实践，打字语法将与函数实现隔离，这将使读取类型签名和实现更容易，因为它们不会相互竞争。这可以作为一个文档大修来实现，同时在Stack Overflow上进行一些宣传。

Conclusion

Translation: 结论

I still love a lot of things about TypeScript, and I'm still hopeful that it improves. Some of these cost concerns may be adequately addressed in the future by adding new features and improving documentation.

Translation: 我仍然喜欢TypeScript的很多东西，我仍然希望它能有所改进。其中一些成本问题可能会在未来通过添加新功能和改进文档得到充分解决。

However, we shouldn't brush these problems under the rug, and it's irresponsible for developers to overstate the benefits of TypeScript without addressing the costs.

Translation: 然而，我们不应该把这些问题掩盖起来，开发人员在不解决成本的情况下夸大TypeScript的好处是不负责任的。

TypeScript can and should get better at type inference, higher order functions, and generics. The TypeScript team also has a huge opportunity to improve documentation, including tutorials, videos, best practices, and an easy-to-find rundown of TypeScript's limitations, which will help TypeScript developers save a lot of time and significantly reduce the costs of using TypeScript.

Translation: TypeScript可以而且应该在类型推理、高阶函数和泛型方面做得更好。TypeScript团队还有一个巨大的机会来改进文档，包括教程、视频、最佳实践，以及易于查找的TypeScript限制概述，这将帮助TypeScript开发人员节省大量时间，并显著降低使用TypeScript的成本。

I'm hopeful that as TypeScript continues to grow, more of its users will get past the honeymoon phase and realize its costs and current limitations. With more users, more great minds can focus on solutions.

Translation: 我希望随着TypeScript的持续增长，它的更多用户将度过蜜月期，并意识到它的成本和当前的局限性。有了更多的用户，更多优秀的人才可以专注于解决方案。

As TypeScript stands, I would definitely use it again in small open-source libraries, primarily to make life easier for other TypeScript users. But I will not use the current version of TypeScript in my next large scale application, because the larger the project is, the more the costs of using TypeScript compound.

Translation: 就TypeScript而言，我肯定会在小型开源库中再次使用它，主要是为了让其他TypeScript用户的生活更轻松。但我不会在下一个大规模应用程序中使用当前版本的TypeScript，因为项目越大，使用TypeScript复合的成本就越高。

This conclusion is ironic because the TypeScript tagline is “JavaScript that Scales”. A more honest tagline might add a word: “JavaScript that scales awkwardly.”

Translation: 这个结论具有讽刺意味，因为TypeScript的口号是“缩放的JavaScript”。一句更诚实的口号可能会加上一个词：“JavaScript伸缩笨拙。”

Updated: 2023–0310 — Added a link to a 2022 study that analyzed over 600 GitHub repositories and discovered that TypeScript did not seem to make a significant difference in bugs. Removed some confusing wording about “theoretical maximum benefit of TypeScript” and instead just emphasized that only ~20% of public bugs on GitHub are addressable by TypeScript.

Translation: 更新时间：2023–0310——添加了2022年一项研究的链接，该研究分析了600多个GitHub存储库，发现TypeScript似乎对bug没有显著影响。删除了一些关于“TypeScript的理论最大效益”的令人困惑的措辞，而只是强调GitHub上只有大约20%的公共漏洞可以通过TypeScript解决。

Updated: 2019–02–11 — Clarified how Airbnb could report a 38% reduction in bugs after adding TypeScript without invalidating any of the claims made in this article. TL;DR — better than average design/spec review + lower than average test coverage could lead to a higher than average ts-addressable % of remaining bugs, without substantially changing the total percentage of bugs prevented by all measures. Updated: Jan 26, 2019 — added “Refactoring” benefit. Updated: Jan 26, 2019 — clarified bug reduction math, increased bug reduction benefit from 8% of remaining bugs to 20% to give TypeScript the maximum benefit of the doubt and demonstrate that it still doesn’t materially impact ROI.

Translation: 更新时间：2019年2月11日-澄清了Airbnb如何在添加TypeScript后报告错误减少38%，而不会使本文中的任何声明无效。TL；DR——优于平均水平的设计/规范审查+低于平均水平的测试覆盖率可能导致剩余错误的可寻址百分比高于平均水平，而不会显著改变所有措施防止的错误的总百分比。

Eric Elliott is a distributed systems expert and author of the books, “Composing Software” and “Programming JavaScript Applications”. As co-founder of DevAnywhere.io, he teaches developers the skills they need to work remotely and embrace work/life balance. He builds and advises development teams for crypto projects, and has contributed to software experiences for Adobe Systems,Zumba Fitness, The Wall Street Journal, ESPN, BBC, and top recording artists including Usher, Frank Ocean, Metallica, and many more.

Translation: Eric Elliott是分布式系统专家，著有《编写软件》和《编程JavaScript应用程序》两本书。作为DevAnywhere.io的联合创始人，他向开发人员传授远程工作和实现工作/生活平衡所需的技能。他为加密货币项目的开发团队提供构建和建议，并为Adobe Systems、Zumba Fitness、《华尔街日报》、ESPN、BBC以及包括Usher、Frank Ocean、Metallica等在内的顶级录音艺术家提供软件体验。

He enjoys a remote lifestyle with the most beautiful woman in the world.

Translation: 他和世界上最美丽的女人过着与世隔绝的生活。